Purpose

This documentation explains the purpose of the DCWorkflow product and how to make use of it. DCWorkflow is a product made to be used with Zope CMF from Zope Corporation. Familiarity with Zope and CMF is assumed.

What is Workflow?

Workflow is the series of interactions that should happen to complete a task. Business organizations have many kinds of workflow. For example, insurance companies process claims, delivery companies track shipments, and schools accept applications for admission. All these tasks involve several people, sometimes take a long time, and vary significantly from organization to organization.

The goal of workflow software is to streamline and track workflow activity. Since different organizations have different workflow processes, workflow software must be flexible and easy to customize.

The DCWorkflow Concept

DCWorkflow makes a few simple assumptions about your workflow:

- There is a single object in the system that represents the task to be completed.
- Every object of a given type goes through the same workflow.
- Tasks are assigned to user roles, not individuals.

DCWorkflow makes it easy to implement workflows that fit this description. If your workflow does not fit these criteria, you should weigh the alternatives. OpenFlow, also for Zope, is more flexible but more complex to set up.

Defining Workflow States

Using workflow states you can add state to your content that is specific to your business process.

CMF comes with a default workflow with three states: private, pending, and published. In the default configuration, all content in the CMF is set to operate in that workflow. When an object is in the private state, only the user who created it and site managers can view and change it. The user is provided with a link to "submit" the content, which puts it in the pending state. Then a user with the "reviewer" role is given the opportunity to either publish or reject the submission, which moves the content object to either the published or private state.

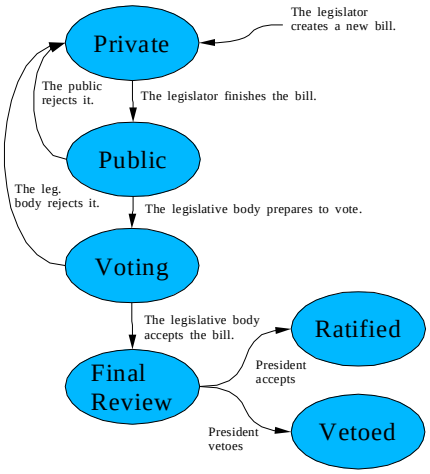

Illustration 1: A Simple Example State Machine

Your business process will most likely require a different set of states. For example, a workflow could naturally model the process of ratifying a bill in a state legislature. You could start by creating a *Private* state to be used while the author is creating a new bill, a *Public* state used before voting, a *Voting* state during which time

legislators are allowed to cast their votes, a *Final Review* state which gives the executive branch time to review it, a *Vetoed* state, and a *Passed* state. (See Illustration 1.)

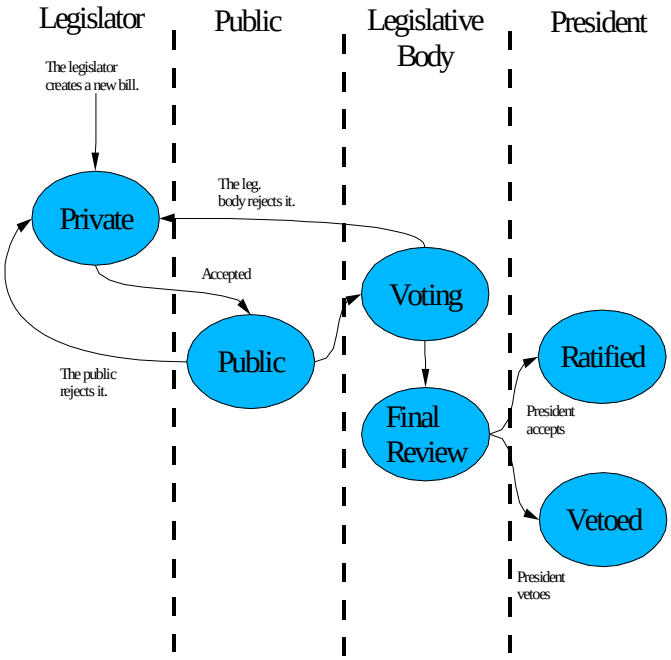

Illustration 2: Using "Swim Lanes" in Designing a Workflow

One good way to determine what states you need in your business process is to first draw a diagram with "swim lanes". (See Illustration 2.) Draw a diagram with each relevant user role at the head of a column, then draw dotted lines between the columns. In the legislature example, the roles might be *Owner*, *Public*, *Senator*, and *President*. Draw a state diagram that shows the flow of your content (in the example, a bill) between the different users. Then create a workflow state for each bubble you draw in your diagram. (See Illustrations 3 and 4.)

A technical note: changing the workflow state of an object does not move it to a different location or add Python attributes to the object. Instead, it asks the workflow tool to set the workflow state of the object and the workflow tool can choose how the state will be stored. The default implementation of the workflow tool stores the workflow state in the `workflow_history` attribute of CMF content objects.

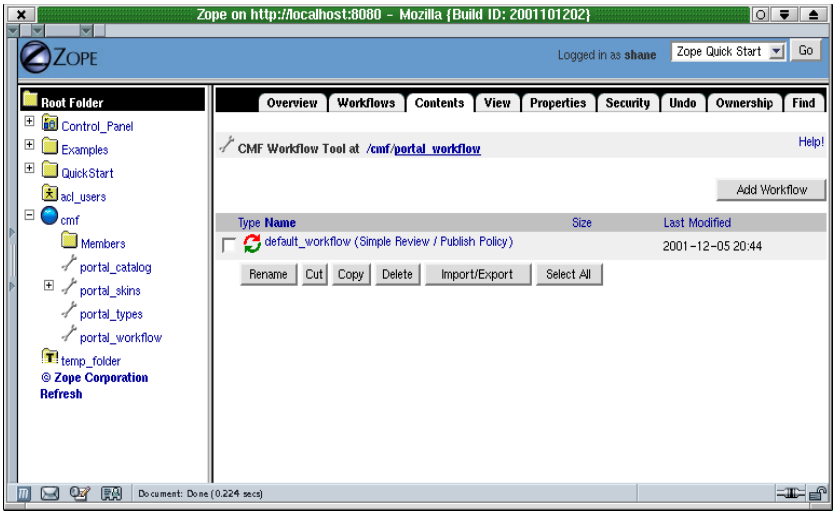

Illustration 3: Add a workflow by clicking the "Add Workflow" button in the portal_workflow tool.

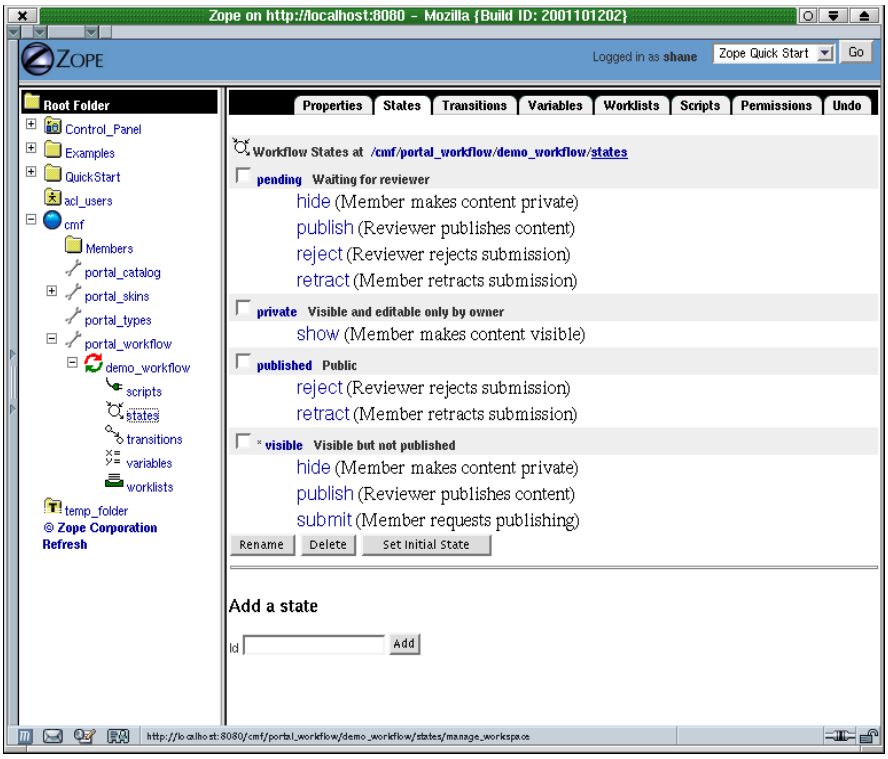

Illustration 4: Creating States

Defining Workflow Transitions

Transitions are the arrows in a state diagram. Generally, for every arrow you draw in your state diagram, create a transition. Some state diagrams, however, have a lot of

arrows pointing to a single state. In that case it might be better to create just one transition that you can reuse in modeling most of the transitions that lead to that state.

Each transition requires a destination state. Select the destination state from the drop-down box. (See Illustration 5 below.)

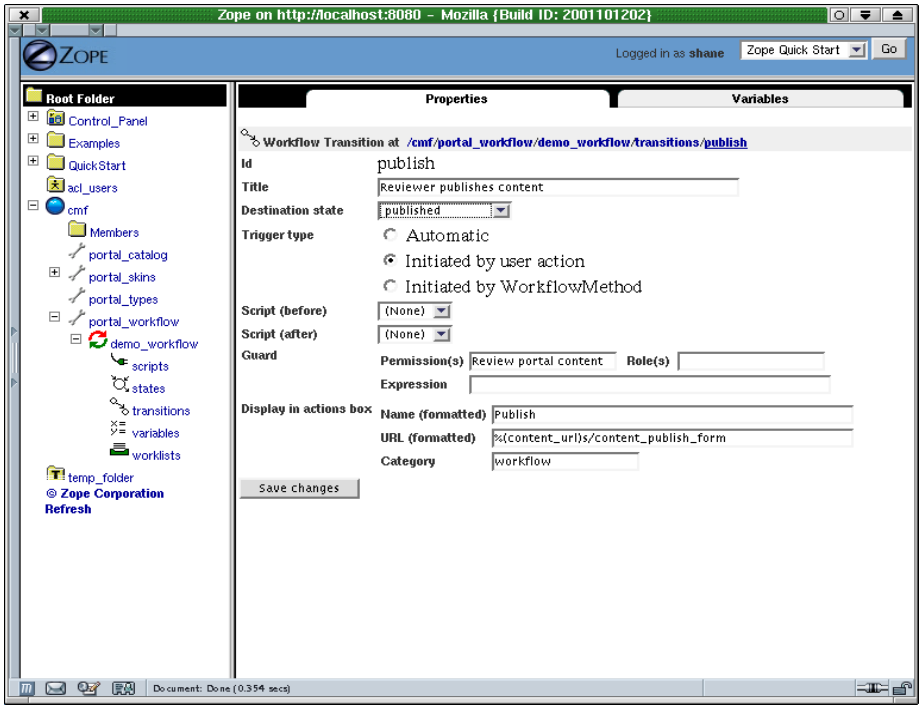

Illustration 5: Configuring a Transition

Transitions are usually protected by a guard condition. If you drew swimming lanes as suggested in Illustration 2, notice that many of the arrows cross the swimming lanes. Every time you cross a swimming lane you need a guard condition. A guard condition can be a permission, a role, or a workflow expression (described later). Guard conditions ensure that only users with the required permission, role, or other criteria can move the object to the new state.

Most transitions are initiated by a user action. For each transition initiated by a user action, enter the information for the corresponding link that should be displayed to the user in the actions box. The link will only be displayed when the user would be

allowed to perform the transition.

Sometimes you need special states and transitions in your workflow diagram that model actions performed in the background, not by any user. In that case you might need to set up transitions that are initiated automatically. Zope will proceed through automatic transitions whenever the guard condition allows it.

Once you have defined all the transitions, go back to your workflow states and define which transitions are allowed to leave those states.

Defining Variables

Often a simple flow of states can't model all the details of a business process. For example, in the bill-passing example, a bill might be allowed to be revised and resubmitted once it is vetoed, but only if it has been vetoed once. If it is vetoed a second time, it is killed for good. To model this behavior, the state machine needs to carry a bit of extra information that "remembers" the past veto.

A variable is a piece of information that transcends states. Most variables are persistent. A variable might hold a counter, a flag, the name of the last user who did some action, or any other simple object.

Variables also serve the purpose of exposing metadata to the catalog or the user. There are five variables in the CMF default workflow: `actor`, `action`, `comments`, `time`, and `review_history`. The first four are there to keep a record of who executed the last transition, what they did, why, and when. The last variable makes it possible for the user to view the workflow history of an object. (See Illustration 6.)

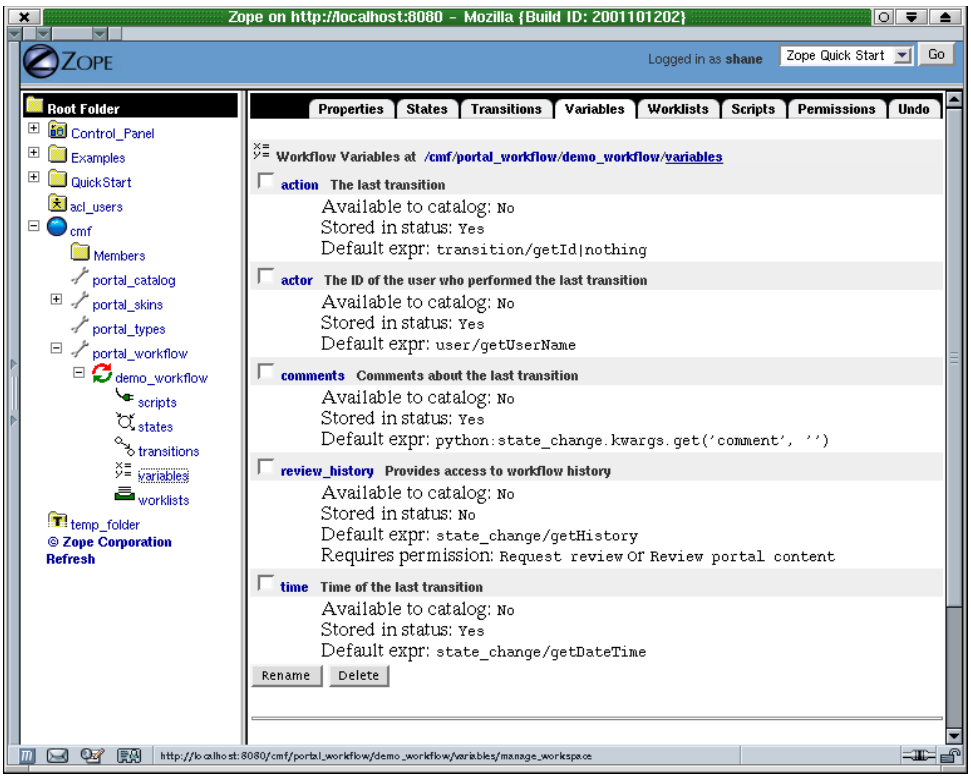

Illustration 6: The Standard Workflow Variables in CMF

You can change the value of a variable when executing a transition or when entering a state. Visit a state you created previously then switch to the *Variables* tab. Here you can reset variables to some specific value. Then visit a transition and switch to the *Variables* tab. Here you can enter expressions that will be computed to determine the new value of a variable.

Defining Worklists

Some users need notification of work that needs to be completed. For example, in the CMF default workflow, users with the *Reviewer* role need to be told when there are items pending review, so they can visit them and either publish or reject them.

One way to accomplish this is with worklists. Worklists add links to users' actions box when there are items in a certain state. The CMF default workflow supplies one

worklist. It shows the *Pending Review* link to reviewers when there are items they are allowed to review.

After creating a worklist, enter the name of the state it matches. Use a guard condition to make only certain users see it. Also enter the name and URL of the link to be displayed.

Defining Scripts

You may need to perform actions like sending an email or invoking another workflow when users execute specific transitions. You can do this by writing scripts. Scripts can be any Zope object, but *Scripts (Python)* are most likely the best choice.

Scripts are passed one parameter, the `state_change` object of a workflow expression. Remember that your script is executed with the permissions of the user who invokes the transition, rather than your own permissions, unless you give the script proxy roles.

Once a script is in the *Scripts* container of a workflow you can visit a transition and select the script to be executed. The script will be executed before the state change and before variables are updated. It can raise exceptions to veto the action or an `ObjectMoved` or `ObjectDeleted` exception to tell the workflow that the object has moved or has been deleted.

Defining Permissions

In a business process, the workflow state of an object usually affects who is allowed to perform non-workflow actions on the object. For example, in the CMF default workflow, the owner of a piece of content is allowed to edit it when it is in the private state, but not when it is in the published state. Anonymous users are allowed to see a

piece of content only if it is in the published state.

CMF accomplishes this by updating the role to permission mappings for objects based on their workflow state. To do this in your own workflow, first determine which permissions should be managed by your workflow by selecting them using the workflow *Permissions* tab.

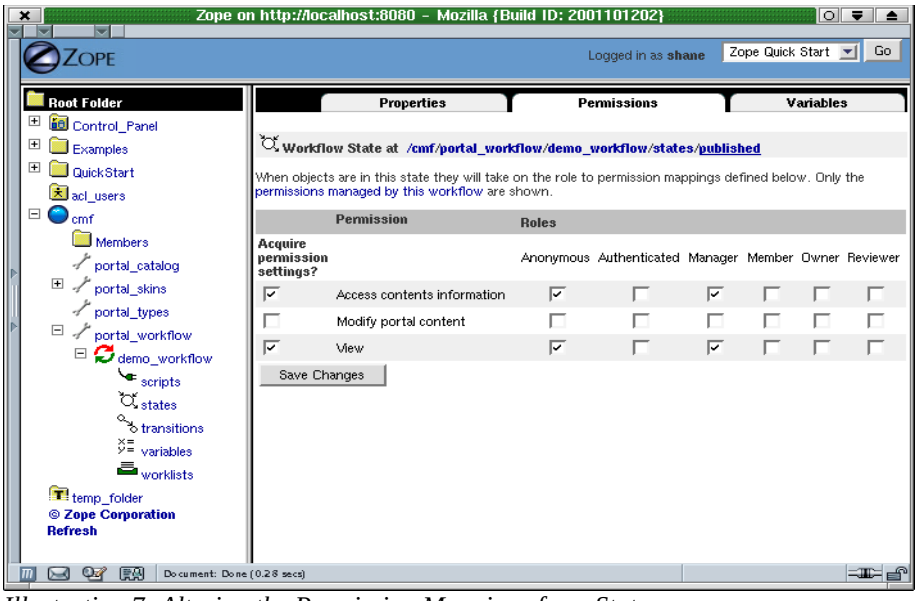

Illustration 7: Altering the Permission Mappings for a State

Then visit the *Permissions* tab of each state and select which roles should have which permissions. This screen is very similar to the familiar *Security* tab. (See Illustration 7.) Remember to turn off the *Acquire Permission* checkbox as necessary.

The role to permission mappings are stored on the objects themselves because that is where they Zope has always stored them. Unfortunately this means that when you change the role to permission mappings in the workflow you need to make sure the changes are applied to the content objects throughout the system. But there is an easy workaround: the default `portal_workflow` tool has a button you can click to update the role to permission mappings in all content objects.

Workflow Expressions

A workflow expression is a TALES expression. TALES expressions are fully described at the following Internet URL:

http://dev.zope.org/Wikis/DevSite/Projects/ZPT/TALES

Some of the contexts have slightly different meanings from what is provided for expressions in page templates.

- `here`: The content object.
- `container`: The content object's container

Several other contexts are also provided.

- `state_change`: A special object containing info about the state change
- `transition`: The transition object being executed
- `status`: The former status
- `workflow`: The workflow definition object
- `scripts`: The scripts in the workflow definition object

`state_change` objects provide the following attributes:

- `status` is a mapping containing the workflow status.
- `object` is the object being modified by workflow.
- `workflow` is the workflow definition object.
- `transition` is the transition object being executed.
- `old_state` is the former state object.
- `new_state` is the destination state object.
- `kwargs` is the keyword arguments passed to the `doActionFor()` method.

- `getHistory()`, a method that returns a copy of the object's workflow history.
- `getPortal()`, which returns the root of the portal.
- `ObjectDeleted` and `ObjectMoved`, exceptions that can be raised by scripts to indicate to the workflow that an object has been moved or deleted.
- `getDateTime()` is a method that returns the `DateTime` of the transition.

Conclusion

DCWorkflow is a valuable tool for its purpose. It lets you create and customize simple workflows for many kinds of tasks. It is not a complete workflow engine, but for the things it was designed to do, it should serve you well.